AUTONOMOUS ADAPTIVE SOFTWARE LOADING FOR A DATA COLLECTION DEVICE

TECHNICAL FIELD

The present disclosure relates generally to software installation, and more particularly but not exclusively, relates to automated techniques to install customized software features into a device, such as a data collection device, without requiring a rebuild of the device's operating system, wherein the software features may be loaded into the device from a wireless network, wired network, local file system, or other location.

BACKGROUND INFORMATION

There are many types of electronic devices. Included among these types of devices are portable electronic devices that have functionalities similar to those of desktop computers. For example, such electronic devices may have a memory, processor, display screen, keyboard, installed software applications, communication components, and so forth. Moreover, such electronic devices may also include an operating system.

In certain situations, electronic devices can be customized differently according to the respective needs of their users. For instance, a group of users may require their electronic devices to have certain software (or hardware) features, while another group of users having the same type of electronic devices may require their electronic devices to have some different (or completely different) software features. This customized configuration of many individual electronic devices can be a tedious and expensive process, particularly in manufacturing situations where "time is money" with regards to mass-producing electronic devices to consumers in a timely manner.

As an example, one technique for customization is to install all possible software features into every electronic device, and then allow consumers

to use only the specific software features that they need, while other installed software features remain installed but unused. In effect, this is really not customization of an electronic device, and moreover, installing all possible software features in electronic devices is an inefficient approach.

Additionally, existing techniques for customized configuration (including software installation) involve manual human intervention to configure each electronic device. Manual configuration is tedious and time consuming, and thus ultimately costly.

Various customization products further illustrate these points. For example, some electronic devices use the Windows® CE operating system from Microsoft Corporation of Redmond, Washington. Windows® CE Platform Builder and WCELoad.exe are two products that allow customization of Windows® CE devices. Platform Builder allows for extensive customization of a Windows® CE device before an operating system image has been created, but no customization afterwards. Thus, if subsequent additional customization is needed, the operating system has to be rebuilt.

This approach is inflexible, since it does not allow existing capabilities to be extended. That is, this approach forces manufactures to install all possible software features into an electronic device, and does not allow for other customization changes that were not known ahead of time when the operating system image was created.

In comparison, WCELoad.exe will install cabinet (CAB) files that have been created by parties offering software, but these CAB files must be executed on the electronic device before customization can be performed. Use of WCELoad.exe allows for customization after the operating system image has been created and is running, but some user intervention is required to perform the customization, including verifying that the installed software is operating properly.

BRIEF SUMMARY OF THE INVENTION

One aspect provides a method to install software features into an electronic device. The method includes storing at least one product configuration matrix (PCM) in the electronic device. The PCM including information is representative of at least one software feature that can be installed in the electronic device. The PCM information is read, and compared with information from a configuration control file (CCF). For a match between the PCM information and the CCF information, the method obtains a software feature that corresponds to the match and installs that software feature into the electronic device.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

Non-limiting and non-exhaustive embodiments are described with reference to the following figures, wherein like reference numerals refer to like parts throughout the various views unless otherwise specified.

Figure 1 is a block diagram of an embodiment of a device with which an embodiment may be implemented.

Figure 2 is a block diagram depicting embodiments of various techniques to provide a configuration control file to the device of Figure 1.

Figure 3 is a flowchart of an embodiment of a technique to install software features into the device of Figure 1.

Figure 4 is a flowchart of an embodiment of a boot sequence that includes the installation technique of Figure 3.

DETAILED DESCRIPTION

Embodiments of techniques for software loading into a device, such as a data collection device, are described herein. In the following description, numerous specific details are given to provide a thorough understanding of embodiments. One skilled in the relevant art will recognize, however, that the invention can be practiced without one or more of the specific details, or with other

methods, components, materials, etc. In other instances, well-known structures, materials, or operations are not shown or described in detail to avoid obscuring aspects of the invention.

Reference throughout this specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment. Thus, the appearances of the phrases "in one embodiment" or "in an embodiment" in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments

As an overview, an embodiment provides a technique to determine which particular software features should be installed in a device without having to rebuild the underlying operating system of the device each time a software feature has to be added, modified, updated, or removed. In one example implementation, the device is a data collection device, although it is appreciated that principles described herein may have application towards other types of devices that require customized software installs and which would benefit from not requiring a rebuild of their operating systems.

The device stores at least one product configuration matrix (PCM) that provides identification information, including which software features can be or should be installed in the device. In one embodiment, the product configuration matrix comprises an alphanumeric string, wherein at least some alphanumeric character(s) or group of characters correspond to a software feature that can be loaded into or otherwise installed into the device.

One or more externally stored configuration control files (CCF), in the form of text files, for example, are provided. The CCF can be stored at any suitable location, such as in a local file system or on a network (wired or wireless). The CCF can contain a description of most, if not all, of the software features that

can be installed in any device. The description includes, for example, product configuration masks, identifications of external locations where software features (*e.g.*, their code) can be obtained, and identifications of locations (*e.g.*, registers) within the device where the software feature is to be loaded. As new features are created or as existing features are modified, the descriptions in the CCF can be updated as needed. Thus, the CCF can be roughly regarded as a type of "master list" of available software features.

During an installation process, the PCM is read from the device and compared with the product configuration masks in the CCF. When there is a match between one or more product configuration masks and the PCM, the software features corresponding to the matching product configuration masks are obtained (such as via network access) and loaded into the device. In an embodiment, the reading of the PCM, comparison between the PCM and the production configuration mask(s), and the obtaining and installation of the software feature(s) are performed during a boot sequence.

Figure 1 is a block diagram of an embodiment of a device 100 with which an embodiment may be implemented. A non-limiting example of the device 100 is the CK30 mobile data collection terminal available from Intermec Corporation of Everett, Washington. Embodiments are applicable to other devices, such as pocket personal computers (PCs), set top boxes, mobile electronic devices, and other consumer electronic devices.

The device 100 includes one or more processors 102, a machine-readable storage medium 104 (such as a system memory), and a system bus 106 that couples various system components, including the storage medium 104, to the processor 102. The processor 102 may be any logic processing unit, such as one or more central processing units (CPUs), digital signal processors (DSPs), application-specific integrated circuits (ASICs), etc.

The system bus 106 can employ any suitable bus structure or architecture, including a memory bus with memory controller, a peripheral bus, and a

local bus. The storage medium 104 includes read-only memory (ROM) 108 and random access memory (RAM) 110. A basic input/output system (BIOS) 112, which can form part of the ROM 108, contains routines that help transfer information between components within the device 100, such as during start-up.

Program modules can be stored in the storage medium 104, such as an operating system 112, one or more application programs 114, other programs or modules 116, and program data 118. An example operating system 112 for the device 100 is Windows® CE, available from Microsoft Corporation of Redmond, Washington. In an embodiment, the operating system 112 can be customized for a particular type of device (sometimes referred to as a "platform") in which the operation system 112 is to be installed. In the context of Windows® CE, this customization is generally referred to as generating an operating system "image" for the platform.

The storage medium 104 may also include a browser 120 for permitting the device 100 to access and exchange data with sources such as web sites of the Internet, corporate intranets, or other networks as described below, as well as other server applications on server computers. The browser 120 in the depicted embodiment is markup language based, such as Hypertext Markup Language (HTML), Extensible Markup Language (XML) or Wireless Markup Language (WML), and operates with markup languages that use syntactically delimited characters added to the data of a document to represent the structure of the document. A number of Web clients or browsers are commercially available such as NETSCAPE NAVIGATOR® from America Online, and INTERNET EXPLORER® available from Microsoft Corporation of Redmond, Washington. While shown in Figure 1 as being stored in the storage medium 104, the operating system 112, application programs 114, other programs/modules 116, program data 118, browser 120, and/or other elements can be stored elsewhere in the device 100.

In an embodiment, the device 100 stores at least one product configuration matrix (PCM) 122. The PCM 122 can be stored in the storage medium 104 or elsewhere in the device 100. In an example implementation, the PCM 122 is

stored in FLASH memory, and can be replaced or updated as needed by re-writing into the FLASH memory.

The PCM 122 of an embodiment comprises an alphanumeric string that contains identifying information associated with the device 100. For instance, the PCM 122 can comprise the following string: CK30CA1214001804. The first four characters (the characters "CK30") can identify the type of device 100. The subsequent alphanumeric characters can identify a version or model number, followed by identification of software features that can be configured in the device 100. As purely by way of example and illustration, the number "8" in the 14th position in the alphanumeric string of the PCM 122 can indicate that the device 100 is to be configured with Japanese text/language user interface software. Other examples of software features include terminal emulation specifications, device driver types, user interface formats, and other features that would be familiar to a person skilled in the art having the benefit of this disclosure.

The individual characters and/or groups of characters in the alphanumeric string of the PCM 122 can, therefore, be used to identify certain software features that are supported by and can be loaded into each individual device 100, which may or may not be the same software features that are supported/loaded in other devices 100. The software features that are ultimately and actually loaded into the device 100 need not necessarily be the entire set of software features that are identified in the PCM 122--the software features can be selected to match the needs of a particular user.

The device 100 may include one or more drives 124 for reading from and writing to a storage unit, such as a floppy disk, compact disk (CD), magnetic cassette, flash memory card, digital video disk (DVD), Bernoulli cartridge, smart cards, and the like. The drive 124 communicates with the processor 102 via the system bus 106. The drive 124 may include interfaces or controllers (not shown) coupled between such drive(s) and the system bus 106. In one embodiment, the drive 124 and/or other drives are not integrated within a housing of the device 100 itself, but instead are external units that are accessible via hardwire or wireless communication interfaces.

A user can enter commands and information into the device 100 through input devices, such as a keyboard 126 (or other input device, such as a mouse, a touch pad, buttons, microphone, joystick, and the like). In an embodiment, the device 100 can include a reader unit 128 as another component to provide information to the device 100. An example of the reader unit 128 is a bar code scanner, an imager, and the like, which can be wireless-based, optical-based, RFID-based, and so forth. The device 100 can include a display 130, which is usable to display user interfaces, menus, graphics, and many types of data.

The device 100 can operate in a networked environment using logical connections to one or more remote computers and/or devices, such as servers, printers, external file systems, devices accessible via wireless or wired networks, and so on. A communication interface 132 is provided to facilitate communication with such devices. The communication interface 132 can comprise a modem, transceiver, Ethernet connection, serial or parallel port connection, Universal Serial Bus (USB) port, and so on.

Figure 2 is a block diagram depicting embodiments of various techniques to provide a configuration control file (CCF) 200 to the device 100 of Figure 1. In an embodiment depicted in Figure 2, the CCF 200 comprises a file that has the following format:

"product configuration mask", "copy from location", "copy to location".

The CCF 200 can be a text file or other file that includes this descriptive information related to software features that can be installed into devices 100. In particular, the product configuration mask appears as an alphanumeric string, such as the following: ******1*********. The asterisks are ignored, but slots (or positions) in the product configuration mask that contain an alphanumeric character are compared against the PCM 122 of the device 100, such as the CK30CA1214001804 PCM from the example above. During the installation process when this comparison is performed, a matching "1" will be found in the seventh position in both the product configuration mask and in the PCM 122.

When a match is found, the "copy from location" field associated with that "1" in the seventh position is consulted to determine where to obtain the code for the corresponding software feature. The "copy to location" field specifies the register or other location in the device 100 where that code is to be loaded.

If a product configuration mask contains all asterisks, then the copy operation will always be performed (*e.g.*, there is a match of all positions in the product configuration mask). Multiple positions in any given product configuration mask can be compared to the PCM 122 if the mask specifies more than one area. For instance, the product configuration mask ****C*1********* would also be considered a match with the example CK30CA1214001804 PCM, but the product configuration mask ****A*1********* would not. In this illustration (assuming the "A" and "C" characters refer to a device version), therefore, the software feature associated with the character "1" for any device having a version "C" will be loaded into the device 100 (which is a CK30 device version "C" according to its PCM 122), instead of loading a software feature associated with the character "1" for a device version A.

There are many ways to organize the correlation between PCMs 122, CCFs 200, and product configuration masks. For example, a single CCF 200 can be created to apply to only a single individual or particular group of devices 100 (such as devices 100 for a particular corporate customer). A single CCF 200 can be created in a manner that it contains multiple product configuration masks that are applicable to either or both just an individual customer or multiple customers. Multiple CCFs 200 may also be applicable to any individual device 100. Using any of these techniques, it is thus possible, to allow customized download of software features into different devices 100, without requiring each and every device 100 to be loaded with all possible software features, of which some may be useless or irrelevant to some of the devices 100.

It is also possible to provide each CCF 200 with just one product configuration mask, rather than multiple masks. Multiple software features can be identified either or both by associating these software features to different positions in a

single product configuration masks, by using multiple configuration masks that have different non-asterisk positions for the corresponding software features, or by using a combination thereof.

In Figure 2, the CCFs 200 are depicted as residing in and accessible to the device 100 via a file system 202, a wireless network 204, a wired network 206, or any suitable combination thereof. The file system 202 can include a database, local storage, storage card, or other storage location that can be accessed by the device 100, such as by connecting the communication interface 132 of the device 100 to the file system 202. For instance, the device 100 can be docked or otherwise plugged into a terminal that provides access to the file system 200.

The networks 204-206 can include, but not be limited to, a local area network (LAN), a wide area network (WAN), or the Internet (*e.g.*, Worldwide Web). Such networking environments are well known in wired and wireless enterprise-wide computer networks, intranets, extranets, and the Internet. Other embodiments include other types of communication networks including cellular networks, paging networks, mobile networks, or other communication networks that can use 802.11, GPS, Bluetooth, cellular (TDMA, FDMA, and/or CDMA), and/or other communication standards.

In addition to the CCFs 200, code 208 can be accessed from the file system 202, wireless network 204, and/or wired network 206. The code 208 comprises the underlying code, modules, subroutines, objects, or other machine-readable instructions of the various software features that can be loaded into devices 100 and that correspond to the description in the CCFs 200. The code 208 can comprise all or parts of the application programs 114, other programs or modules 116, and/or program data 118 shown in Figure 1. Such code 208 is located and obtained via the "copy from location" information indicated in the CCFs 200 during the installation process when the PCM 122 is compared to the product configuration mask in the CCF 200. In one embodiment, the code 208 is in the

form of cabinet (CAB) files or other suitable format that encapsulates or packages the code.

Figure 3 is a flowchart 300 of an embodiment of a technique to install software features into the device 100 of Figure 1. At least some of the operations depicted in the flowchart 300 (and other flowcharts shown and described herein) may be embodied in software or other machine-readable instruction stored on a machine-readable medium. For example, the installation may be performed by software on the storage medium 104 during part of a boot sequence, as will be described in further detail with respect to Figure 4. Alternatively or additionally, the installation may be performed by the device 100 outside of the boot sequence. In another embodiment, the installation may be performed by a machine that is external to the device 100, such as by a terminal to which the device 100 is docked during installation.

It is appreciated that the operations depicted in the flowchart 300 (as well as other flowcharts shown and described herein) need not necessarily occur in the exact order shown. Moreover, various operations may be added, removed, modified, or combined in other embodiments.

At a block 302, the installation process starts. In an embodiment, the installation process is referred to as an automatic install ("autoinstall") in that the installation can be performed automatically, such as during part of the boot sequence for the device 100, without user intervention.

At a block 304, the device 100 (or other machine performing the installation) searches for a CCF 200. Such searching can involve making connections to and querying the file system 202, wireless network 204, and/or wired network 206 for the presence of one or more CCFs 200. As discussed above, software upgrades or additions can be made available by updating the CCFs 200 (which can be thought of as a type of "master list"), storing the updated CCFs 200, and storing the code 208 that embodies the software upgrades/additions.

If no CCF 200 is found (or if no applicable CCF 200 is found), then the autoinstall process exits at a block 320, and the next process in the boot sequence is performed. Otherwise if an applicable CCF 200 is found at the block 304, then the PCM 122 is loaded or otherwise read from the device 100 at a block 306. During this reading, the alphanumeric string contained in the PCM 122 is obtained so that it can be compared with the product configuration mask(s) in the CCF 200.

At a block 308, the first product configuration mask is located or otherwise obtained from the CCF 200. As discussed above, there may be several possible ways to arrange product configuration masks and CCFs 200. For instance, a single CCF 200 may contain multiple product configuration masks, with different software features being identified in non-asterisk positions in multiple different product configuration masks. Alternatively or additionally, multiple software features may be identified in non-asterisk positions in individual product configuration masks. Thus, according to various possible embodiments that can be implemented, the searching for and comparing of masks depicted in the flowchart 300 can involve either or both comparison of the PCM 122 to multiple different product configuration masks contained in one or more CCFs 200, or comparison of each alphanumeric character of the PCM 122 to corresponding character positions in only one product configuration mask. A person skilled in the art having the benefit of this disclosure will realize that there are many ways to arrange the encoding of multiple or single masks (including hierarchical and dependent arrangements of masks and software features identified therein) in a manner to account for specific and/or general matches.

If there is no match between the PCM 122 and the current product configuration mask at a block 310, then the next mask is obtained for comparison at a block 314. If there is a match at the block 310, then the appropriate action is performed at a block 312 to load the corresponding software feature into the device 100.

According to an embodiment, the action(s) performed at the block 312 can include the following: parse through or otherwise read the CCF 200 to determine the location of the matching software feature(s) by examining the "copy from location" fields; connect to such locations (which may be in the file system 202, wireless network 204, wired network 206, or other location or combination thereof); download or otherwise obtain the code 208 from such location; and install the downloaded code into the device 100 by using the "copy to location" field in the CCF 200 to determine a register or other location in the device 100 where the code is to be installed.

In an embodiment, the "copying from" and "copying to" operations can involve the copying of CAB files that encapsulate or otherwise package the code 208. It is appreciated that other file formats and/or technique for information transfer may be used, alternatively or additionally to using CAB files.

At a block 316, no more masks are available for comparison. This result may occur, for instance, after all possible software features have been identified and installed, or if no matches have been found in any of the CCFs 200. At a block 320, the autoinstall process is exited. Upon exit of the autoinstall process, subsequent operations associated with the boot sequence can be performed and/or the device 100 is ready for use.

As discussed above, the autoinstall process depicted in Figure 3 can be performed during the boot sequence for the device 100. Performing the installation during the boot sequence is advantageous in that doing so requires little, if any, human/user intervention in the installation and is automatic.

For purposes of illustration and explanation only, below is an example of a boot sequence for a device having Windows® CE as its operating system:

20 – Device.exe

30 – GWES.exe (depends on Device.exe being loaded and ready)

50 – Explorer.exe (depends on Device.exe and GWES.exe being loaded and ready)

60 – Services.exe (depends on Device.exe being loaded and ready).

In this example boot sequence, Device.exe loads the device drivers. GWES.exe loads the graphical and windowing event subsystems. Explorer.exe loads the user interface for the operating system shell, and services.exe is a process that supplements the Device.exe process. A person skilled in the art would be familiar with the features and operations associated with these individual executable processes in the boot sequence, and therefore, such processes will not be described in detail herein. It is appreciated that other types of operating systems may involve processes or executable files in their boot sequences that are not necessarily the same as depicted above—it is understood that embodiments can be provided wherein the autoinstall process can be implemented in these other types of operating systems.

Figure 4 is a flowchart of an embodiment of a boot sequence 400 that includes the automatic installation process depicted by the flowchart 300 of Figure 3. The boot sequence includes the Device.exe and GWES.exe (or similar executables) loading at blocks 402 and 404, respectively. The Explore.exe and Services.exe (or similar executables) are loaded at blocks 406 and 408, respectively. In one embodiment, the autoinstall process depicted by the flowchart 300 is performed subsequent to loading the GWES.exe at the block 404 and prior to loading the Explorer.exe at the block 406.

The autoinstall process is performed at this point in the boot sequence due to several reasons. First, the Device.exe loads the drivers and other components that are used for establishing and maintaining communications between the device 100 and external devices (such as the file system 202 and devices in the networks 204 and 206). These include drivers for modems, Ethernet cards, and the like. By having such communication-related drivers loaded into the device 100 and ready by the time the autoinstall process occurs,

the autoinstall process can utilize such drivers to communicate with and download the code 208 from external locations. Second, Explorer.exe runs terminal emulation processes or other processes related to the user interface(s) for the operating system 112 of the device 100. By performing the autoinstall process prior to loading Explorer.exe, the software features (such as terminal emulation) required by or used by Explorer.exe are guaranteed to be present/installed in the device 100 by the time Explorer.exe loads and runs at the block 406.

While the embodiment of Figure 4 has been described with the autoinstall process occurring during a specific point in the boot sequence, it is appreciated that the autoinstall process may be performed, in other embodiments, at other points in a boot sequence. The specific points to perform the autoinstall process in the boot sequence can be varied based on several possible factors, including but not limited to, the types of software features that are expected to be loaded, the particular executable files that are loaded during the boot sequence, the dependencies among the executable files, and so forth. Moreover, other embodiments can be provided wherein the autoinstall process is performed outside of the boot sequence.

All of the above U.S. patents, U.S. patent application publications, U.S. patent applications, foreign patents, foreign patent applications and non-patent publications referred to in this specification and/or listed in the Application Data Sheet, are incorporated herein by reference, in their entirety.

The above description of illustrated embodiments, including what is described in the Abstract, is not intended to be exhaustive or to limit the invention to the precise forms disclosed. While specific embodiments and examples are described herein for illustrative purposes, various equivalent modifications are possible within the scope of the invention and can be made without deviating from the spirit and scope of the invention.

For example, while an embodiment has been described with respect to a device 100 that uses Microsoft Windows® CE as the operating system 112, it

is appreciated that embodiments may be implemented in conjunction with other types of operating systems. Also, the boot sequence 400 may involve processes that are in addition or alternative to those depicted in Figure 4—the autoinstall process may be implemented in such a boot sequence nevertheless.

These and other modifications can be made to the invention in light of the above detailed description. The terms used in the following claims should not be construed to limit the invention to the specific embodiments disclosed in the specification and the claims. Rather, the scope of the invention is to be determined entirely by the following claims, which are to be construed in accordance with established doctrines of claim interpretation.